El propósito de este documento es especificar las funcionalidades de la aplicación "Canchas", así como las tecnologías y arquitecturas que se utilizarán para implementarlas. El objetivo es proporcionar una visión clara que permita seleccionar las tecnologías más adecuadas para el desarrollo del proyecto. Considerando las dimensiones del proyecto, se espera un alto volumen de peticiones y usuarios simultáneos. Por lo tanto, se busca asegurar una alta disponibilidad y optimización de los servicios.

**Listado de funcionalidades:**

1. Administración de usuarios.
   1. Registro y autenticación de usuarios.
   2. Administración de perfiles.
   3. Automación y roles.
2. Conexión entre jugadores.
   1. Conexión entre jugadores para organizar partidos.
   2. Búsqueda de jugadores y equipos.
   3. Gestión de invitaciones y aceptación de partidos.
3. Gestión de canchas.
   1. Reserva de canchas.
   2. Gestión de disponibilidad.
   3. Pagos y confirmación de reservas.
4. Funcionalidad de notificaciones.
   1. Envió de notificaciones push y correos electrónicos.
   2. Alerta de reservas, partidos y apuestas.
   3. Notificaciones de eventos.
5. Funcionalidad de apuestas.
   1. Gestión de apuestas entre partidos.
   2. Calculo y pago de premios.
   3. Historial de apuestas y transacciones.
6. Funcionalidad de chat.
   1. Búsqueda de usuarios.
   2. Envió de mensajes en tiempo real.

**Tecnologías sugeridas.**

Teniendo en cuenta que la finalidad es tener una plataforma administrativa dedicada a las actividades desempeñadas por los administradores de las canchas sintéticas y una plataforma móvil enfocada a los jugadores amateurs organizar y agendar partidos, se relacionan las tecnologías propuestas para llevar el desarrollo.

|  |  |  |
| --- | --- | --- |
| **Plataforma** | **Tecnología** | **Version** |
| Web | Angular |  |
| Móvil | Flutter |  |
| Backend | Mixta (Microservicios) |  |

**Descripción de arquitecturas.**

**Backend**

Se opta por implementar la arquitectura basada en microservicios para la creación y estructuración backend de la aplicación.

**Microservicios:**

A continuación, se relacionan las tecnologías y motores de bases de datos que se usaran para la creación de cada uno de los servicios, los servicios se definen en base a las funcionalidades anteriormente descritas.

1. Api Geteway.

* Centraliza las peticiones.
* Autentica y autoriza las solicites.
* Distribuye las solicites a cada uno de los servicios
* Tecnología a usar: Spring boot.
* No usa base de datos.

1. User service.

* Tecnología a usar: Spring boot.
* Motor de datos: PostgreSQL

1. Player connection service.

* Tecnología a usar: Node.js
* Motor de datos: MongoDB

1. Field Management service.

* Tecnología a usar: Spring boot.
* Motor de datos: PostgreSQL

1. Notification service.

* Tecnología a usar: Node.js
* Motor de datos: Redis

1. Betting service.

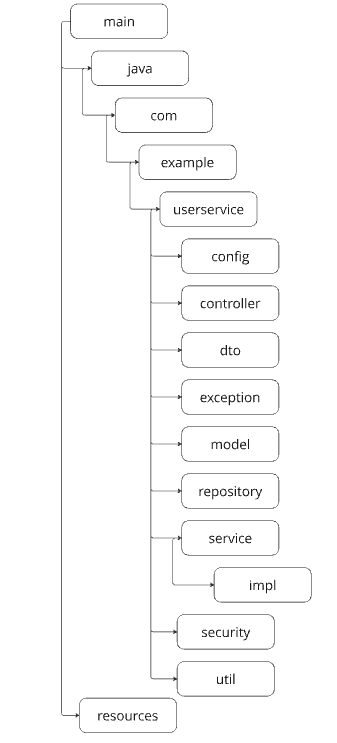
* Tecnología a usar: Spring boot.
* Motor de datos: PostgreSQL.

1. Chat service.

* Tecnología| a usar: Node.js
* Motor de datos: MongoDB.

**Distribución de carpetas Use service.**

A continuación, se relaciona un ejemplo de la distribución de capetas para un servicio con la tecnología spring boot.



* **config/**

Nombre archivo: AppConfig.java

Configuración general de la aplicación, como beans y propiedades específicas.

* **controller/**

Nombre archivo: UserController.java

Controladores REST que manejan las solicitudes HTTP relacionadas con usuarios.

* **dto/**

Nombre archivo: UserDTO.java

Transferencia de datos

* **exception/**

Nombre archivo: GlobalExceptionHandler.java

Maneja todas las excepciones globales de la aplicación

* **model/**

Nombre archivo: User.java

Clases que representan las entidades de la base de datos.

* **repository/**

Nombre archivo: UserRepository.java

Interfaces que extienden JpaRepository o CrudRepository para interactuar con la base de datos.

* **service/**

Nombre archivo: UserService.java

Clases abstractas que definen las funcionalidades.

* **service/impl/**

Nombre archivo: UserServiceImpl.java

Clases que extiende de UserService y contienen la lógica de negocio.

* **security/**

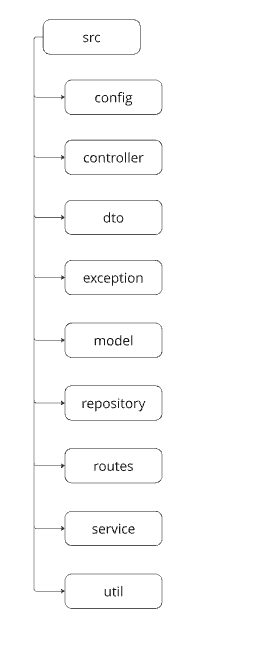
Nombre archivo: SecurityConfig.java

Configuración de seguridad, incluyendo reglas de acceso y protección de rutas.

* **util/J**wtUtil.java

**Distribución de carpetas Player Connection Service.**

A continuación, se relaciona un ejemplo de la distribución de capetas para un servicio con la tecnología Node.js.



* **config/**

Nombre archivo: appConfig.ts

Configuracion general de la aplicación, como conexiones a bases de datos

* **controller/**

Nombre archivo: playerController.ts

Controlador encargado de procesar las solicitudes HTTP

* **dto/**

Nombre archivo: playerDTO.ts

Encargado de transferencia de datos

* **exception/**

Nombre archivo: playerNotFoundException.ts

Excepciones personalizadas-

* **middleware/**

Nombre archivo: authMiddleware.ts

Autenticación y autorización de usuarios

* **model/**

Nombre archivo: player.ts

Representación de las entidades de la base de datos.

* **repository/**

Nombre archivo: playerRepository.ts

Interfaces y clases que para interacción con la base de datos.

* **routes/**

Nombre archivo: playerRoutes.ts

Definición de rutas

* **service/**

Nombre archivo: playerService.ts

Implementación de la lógica de negocio

* **util/**

Nombre archivo: util.ts

Definición de funcionalidades genéricas.

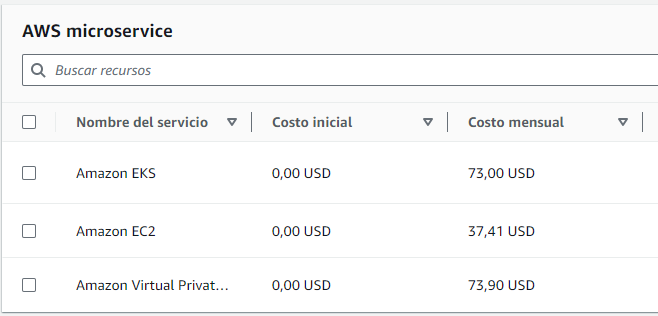
**Costos**

2024-05-08

A continuación, se muestra cotización realizada en los principales prestadores de servicio nube, teniendo en cuenta la arquitectura, tecnologías y servicios anteriormente planteados.

**AWS**

|  |  |  |
| --- | --- | --- |
| **Descripción** | **Valor USD** | **Datos adicionales** |
| Amazon EKS | 73 |  |
| Amazon Virtual Private Cloud (VPC) | 73.90 | 10 GB mes |
| Amazon EC2 | 38.61 | 2 CPU, 8GB RAM, 32GB SSD |
| Total | 185.51 |  |



[**https://calculator.aws/#/estimate?id=793f239a178f55ece007474fb0e61b8a5987a866**](https://calculator.aws/#/estimate?id=793f239a178f55ece007474fb0e61b8a5987a866)

**Amazon EKS:** Servicio administrado que permite realizar la ejecución de kubernetes en la nube de AWS. Facilita la instalación mantenimiento y operación de kubernestes.

**Kubernetes:** Es una plataforma de código abierto que facilita la implementación, escalado y gestión de aplicaciones en contenedores. Permite administrar automáticamente las cargas de trabajo y servicios, lo que optimiza recursos y mejora la portabilidad de las aplicaciones.

**Amazon Virtual Private Cloud (VPC):** Es una red virtual dedicada a la cuenta de aws contratada, ofrece un control total sobre el entorno real de redes virtuales. Esta infraestructura está aislada lógicamente de otras redes permitiendo gestionar la ubicación de los recursos y configuraciones como habitualmente lo hacen los centros de datos.

**Amazon EC2:** Es un servicio web de Amazon que proporciona capacidad de computación en la nube, segura y de tamaño modificable. Permite lanzar instancias de servidores virtuales en la nube, con la flexibilidad de elegir el hardware y la configuración que mejor se adapte a los requerimientos.